

Converting the MPI application to a Hybrid OpenMP/MPI application

Task 2 Parallel Analysis, Scoping and Vectorization

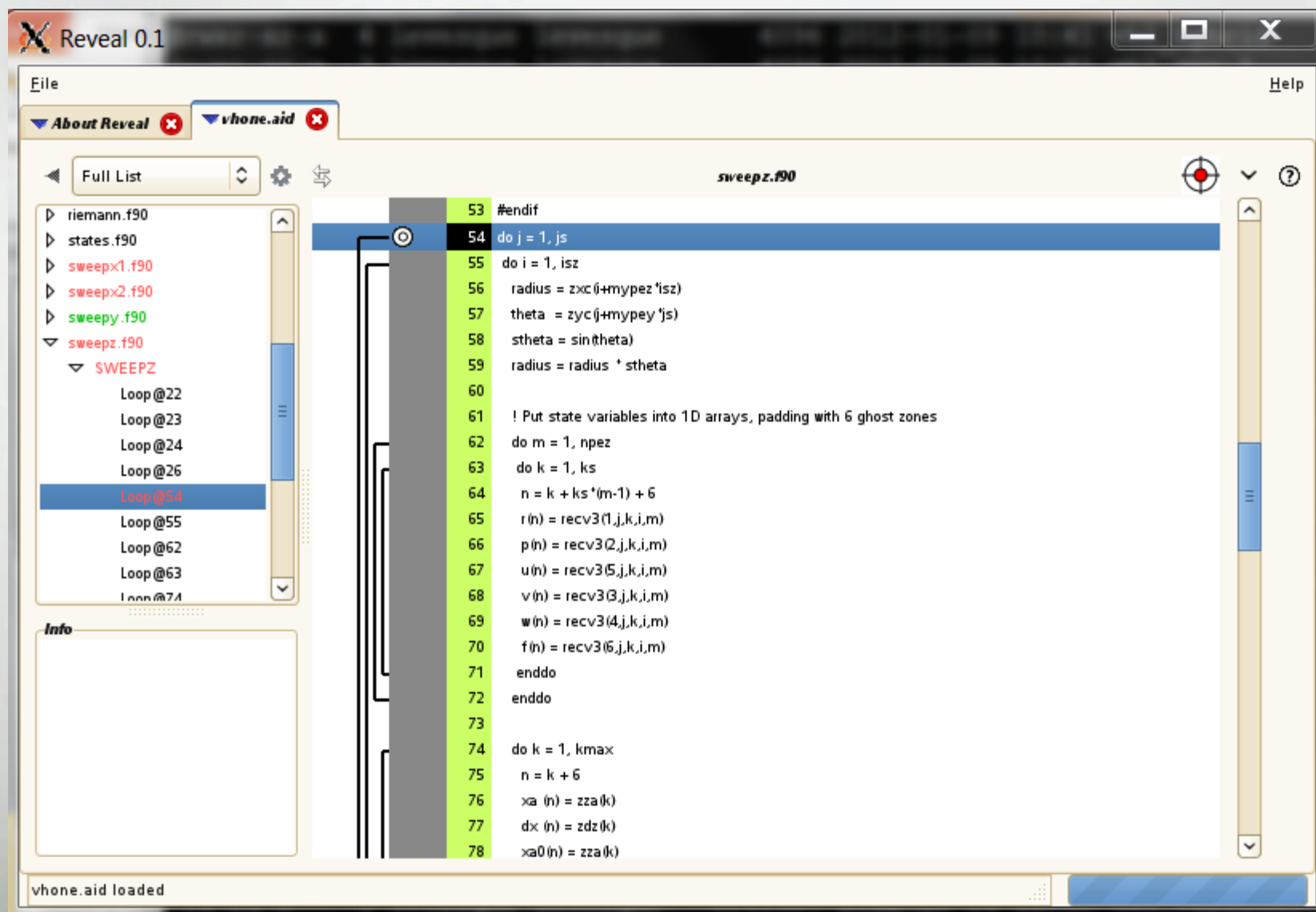
- Investigate parallelizability of high level looping structures
 - Often times one level of loop is not enough, must have several parallel loops
 - User must understand what high level DO loops are in fact independent.
 - Without tools, variable scoping of high level loops is very difficult
 - Loops must be more than independent, their variable usage must adhere to private data local to a thread or global shared across all the threads
- Investigate vectorizability of lower level Do loops
 - Cray compiler has been vectorizing complex codes for over 30 years

Converting the MPI application to a Hybrid OpenMP/MPI application

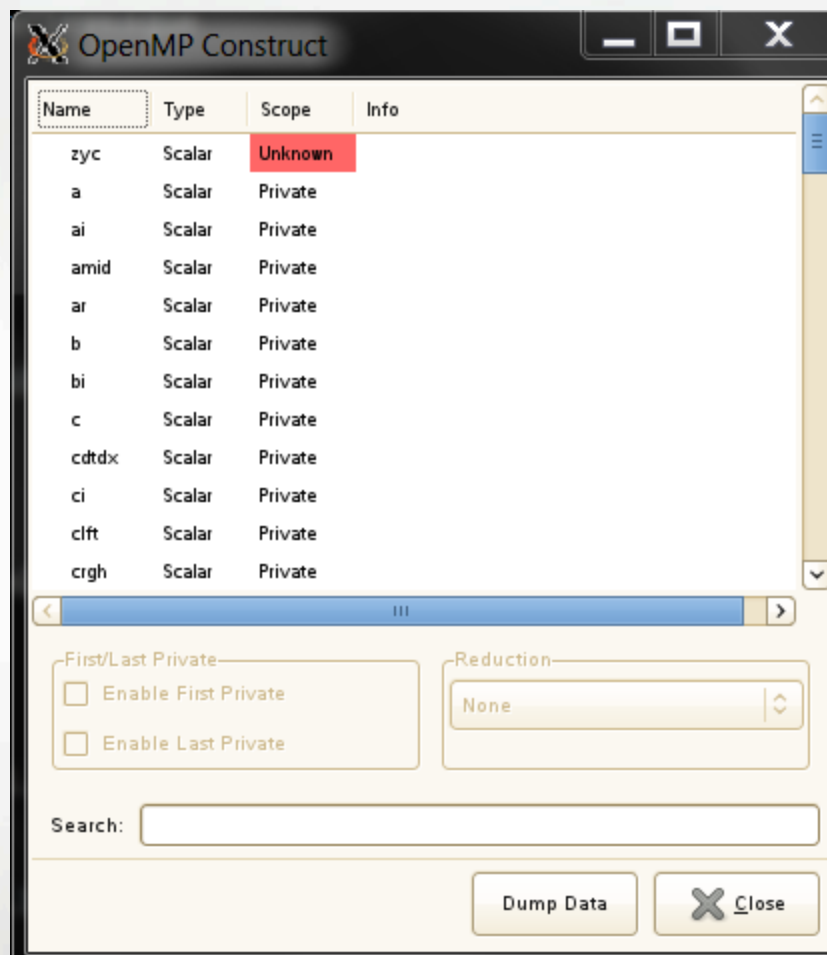
Task 2 Parallel Analysis, Scoping and Vectorization (Cont)

- Current scoping tool, -homp_analyze, is meant to interface to a code restructuring GUI called “reveal”. At this time, we need to use cryptic output and massage it with editor/script.
 - !dir\$ omp_analyze_loop
- In order to utilize scoping tool for loops that contain procedures the program library need to be employed
 - -hwp -hpl=vhone.aid
 - This will do an initial pass of the code, checking for error and then at the load it will build the program library and perform the analysis
- Compiler will be very conservative
 - <object_message kind="warn">LastPrivate of array may be very expensive.</object_message>

Main window of reveal



Scoping window



The image shows a software window titled "OpenMP Construct". It contains a table with four columns: Name, Type, Scope, and Info. The table lists several variables, all of which are Scalars. The "Scope" for the first variable, "zyc", is "Unknown" and is highlighted in red. The other variables have a "Private" scope. Below the table, there are two groups of controls. The first group, labeled "First/Last Private", contains two checkboxes: "Enable First Private" and "Enable Last Private", both of which are currently unchecked. The second group, labeled "Reduction", contains a dropdown menu currently set to "None". At the bottom of the window, there is a "Search:" text box and two buttons: "Dump Data" and "Close".

Name	Type	Scope	Info
zyc	Scalar	Unknown	
a	Scalar	Private	
ai	Scalar	Private	
amid	Scalar	Private	
ar	Scalar	Private	
b	Scalar	Private	
bi	Scalar	Private	
c	Scalar	Private	
cdtdx	Scalar	Private	
ci	Scalar	Private	
clft	Scalar	Private	
crgh	Scalar	Private	

First/Last Private

☐ Enable First Private

☐ Enable Last Private

Reduction

None

Search:

Dump Data Close

At this point we should have some idea of the major arrays

- 1) Which arrays are use in the major computational routines?
- 2) Where else are these arrays used?
- 3) Are other arrays used with identified arrays
- 4) Go to 1

This is extremely difficult in Fortran and more so in C and C++. We could really used a tool that identified where in the code certain range of memory was used.

What we end up finding out

Private Variables in module, need to use Threadprivate

```
!$omp threadprivate (r, p, e, q, u, v, w, xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta,
stheta)
real, dimension(maxsweep) :: r, p, e, q, u, v, w           ! fluid variables
real, dimension(maxsweep) :: xa, xa0, dx, dx0, dvol        ! coordinate values
real, dimension(maxsweep) :: f, flat                      ! flattening parameter
real, dimension(maxsweep,5) :: para                       ! parabolic interpolation
coefficients
real :: radius, theta, stheta
```

Reduction variable down callchain, need to use

!\$OMP CRITICAL;!\$OMP END CRITICAL

```
hdt    = 0.5*dt
do n = nmin-4, nmax+4
  Cdt dx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
enddo
!$omp critical
do n = nmin-4, nmax+4
  svel      = max(svel, Cdt dx (n))
enddo
!$omp end critical
do n = nmin-4, nmax+4
  Cdt dx (n) = Cdt dx (n)*hdt
  fCdt dx (n) = 1. - fourthd*Cdt dx (n)
enddo
```

Task 3 Moving from OpenMP to OpenACC

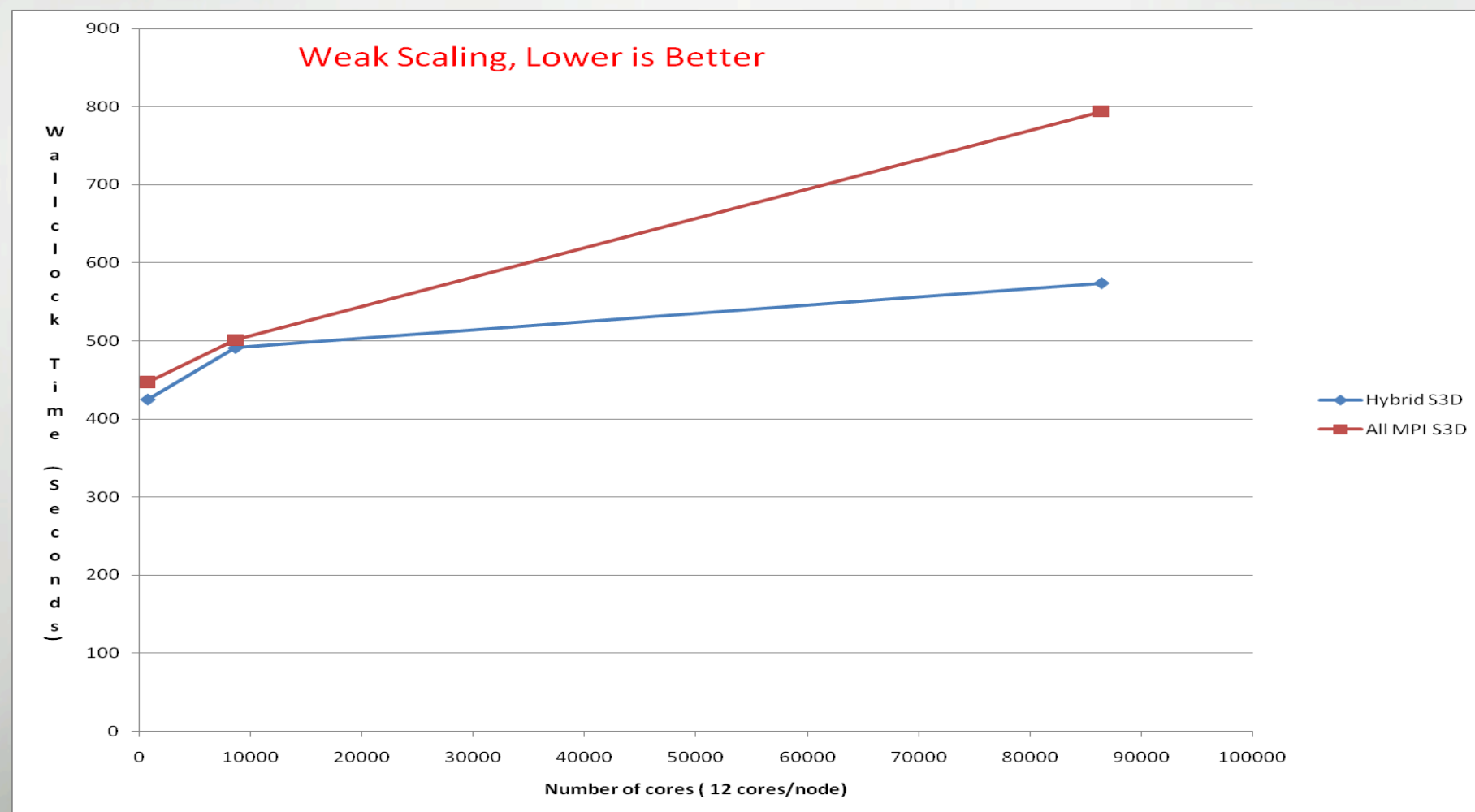
- Things that are different between OpenMP and OpenACC
 - Cannot have CRITICAL REGION down callchain
 - Cannot have THREADPRIVATE
 - Vectorization is much more important
 - Cache/Memory Optimization much more important
 - No EQUIVALENCE
- Currently both OpenMP and OpenACC must be included in the source

```

#ifdef GPU
!$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
!$acc&    xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)&
!$acc&    reduction(max:svel)
#else
!$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
!$omp&    xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)&
!$omp&    reduction(max:svel)
#endif

```

Resultant Hybrid S3D Performance

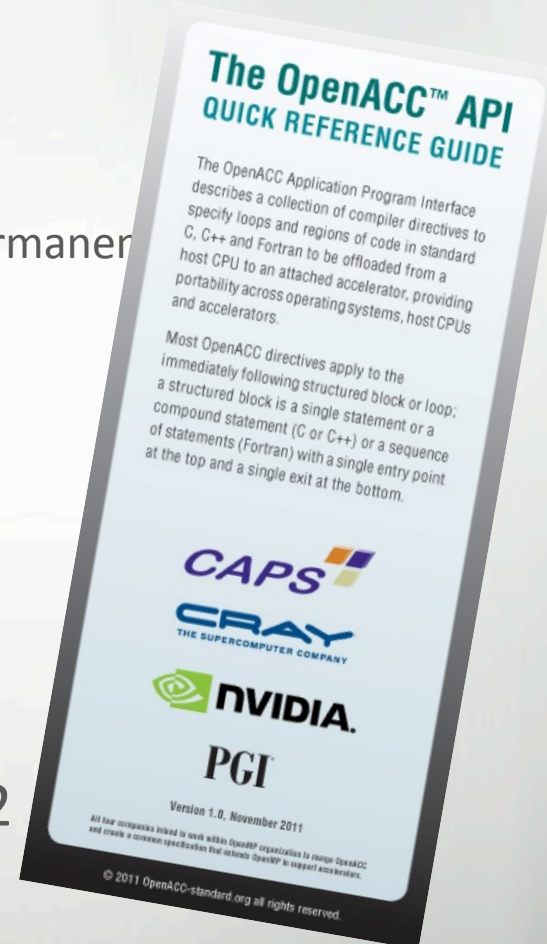


NVIDIA, Cray, PGI, CAPS Unveil 'OpenACC' Programming Standard for Parallel Computing

*Directives-based Programming Makes
Accelerating Applications Using
CPUs and GPUs Dramatically Easier*



- A common directive programming model for today's GPUs
 - Announced at SC11 conference
 - Offers portability between compilers
 - Drawn up by: NVIDIA, Cray, PGI, CAPS
 - Multiple compilers offer portability, debugging, permanent
 - Works for Fortran, C, C++
 - Standard available at www.OpenACC-standard.org
 - Initially implementations targeted at NVIDIA GPUs
- Current version: 1.0 (November 2011)
- Compiler support:
 - Cray CCE: partial now, complete in 2012
 - PGI Accelerator: released product in 2012
 - CAPS: released product in Q1 2012



Using directives to give the compiler information

- Developing efficient OpenMP regions is not an easy task; however, the performance will definitely be worth the effort
- The next step will be to add OpenACC directives to allow for compilation of the same OpenMP regions to accelerator by the compiler.
 - With OpenACC data transfers between multi-core socket and the accelerator as well as utilization of registers and shared memory can be optimized.
 - With OpenACC user can control the utilization of the accelerator memory and functional units.

Task 3 Correctness Debugging

- Run transformed application on the accelerator and investigate the correctness and performance
 - Run as OpenMP application on multi-core socket
 - Use multi-core socket Debugger - DDT
 - Run as Hybrid multi-core application across multi-core socket and accelerator
- Tools That will be needed
 - Information that was supplied by the directives/user's interaction with the compiler

Task 4 Letting the Compiler do all the work

- The only requirement for using the !\$acc parallel loop is that the user specify the private variables and the compiler will do the rest.
 - If subroutine calls are contained in the loop, -hwp must be used.

```
#ifdef GPU
```

```
!$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&  
!$acc&    xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)&  
!$acc&    reduction(max:svel)
```

```
#else
```

```
!$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w, svel0,&  
!$omp&    xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)&  
!$omp&    reduction(max:svel)  
#endif
```

- The Compiler will then show:
 - All data motion required to run the loop on the accelerator.
 - Show how it handled the looping structures in the parallel region

Compiler list for SWEEPX1

```

45.                #ifdef GPU
46.  G-----< !$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
47.  G          !$acc&      xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)&
48.  G          !$acc&      reduction(max:svel)
49.  G          #else
50.  G          !$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
51.  G          !$omp&      xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)&
52.  G          !$omp&      reduction(max:svel)
53.  G          #endif
55.  G g-----< do k = 1, ks
56.  G g 3-----< do j = 1, js
57.  G g 3          theta=0.0
58.  G g 3          stheta=0.0
59.  G g 3          radius=0.0
62.  G g 3 g-----< do i = 1,imax
63.  G g 3 g          n = i + 6
64.  G g 3 g          r (n) = zro(i,j,k)
65.  G g 3 g          p (n) = zpr(i,j,k)
66.  G g 3 g          u (n) = zux(i,j,k)
67.  G g 3 g          v (n) = zuy(i,j,k)
68.  G g 3 g          w (n) = zuz(i,j,k)
69.  G g 3 g          f (n) = zfl(i,j,k)
71.  G g 3 g          xa0(n) = zxa(i)
72.  G g 3 g          dx0(n) = zdx(i)
73.  G g 3 g          xa (n) = zxa(i)
74.  G g 3 g          dx (n) = zdx(i)
75.  G g 3 g          p (n) = max(smallp,p(n))
76.  G g 3 g          e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
77.  G g 3 g-----> enddo
79.  G g 3          ! Do 1D hydro update using PPMLR
80.  G g 3 gr2 I--> call ppmlr (svel0, sweep, nmin, nmax, ngeom, nleft, nright,r, p, e, q, u, v, w, &
81.  G g 3          xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)
82.  G g 3

```

Compiler list for SWEEPX1

ftn-6405 ftn: ACCEL File = sweepx1.f90, Line = 46

A region starting at line 46 and ending at line 104 was placed on the accelerator.

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zro" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zpr" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zux" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zuy" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zuz" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zfl" to accelerator, free at line 104 (acc_copyin).

ftn-6416 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "send1" to accelerator, copy back at line 104 (acc_copy).

Task 5 Fine tuning of accelerated program

- Understand current performance bottlenecks
 - Is data transfer between multi-core socket and accelerator a bottleneck?
 - Is shared memory and registers on the accelerator being used effectively?
 - Is the accelerator code utilizing the MIMD parallel units?
 - Is the shared memory parallelization load balanced?
 - Is the low level accelerator code vectorized?
 - Are the memory accesses effectively utilizing the memory bandwidth?

Profile of Accelerated Version 1

Table 1: Time and Bytes Transferred for Accelerator Regions

Acc Time%	Acc Time	Host Time	Acc Copy In (MBytes)	Acc Copy Out (MBytes)	Calls	Function PE=HIDE Thread=HIDE
100.0%	58.363	67.688	24006.022	16514.196	14007	Total
30.3%	17.697	0.022	--	--	1000	sweepy_.ACC_KERNEL@li.47
22.0%	12.827	0.010	--	--	500	sweepx2_.ACC_KERNEL@li.46
21.2%	12.374	0.013	--	--	500	sweepz_.ACC_KERNEL@li.67
14.0%	8.170	0.013	--	--	500	sweepx1_.ACC_KERNEL@li.46
3.9%	2.281	1.161	12000.004	--	1000	sweepy_.ACC_COPY@li.47
2.0%	1.162	0.601	6000.002	--	500	sweepz_.ACC_COPY@li.67
1.6%	0.953	0.014	--	6000.004	1000	sweepy_.ACC_COPY@li.129
1.0%	0.593	0.546	3000.002	--	500	sweepx1_.ACC_COPY@li.46
1.0%	0.591	0.533	3000.002	--	500	sweepx2_.ACC_COPY@li.46
0.8%	0.494	0.015	--	3000.002	500	sweepx2_.ACC_COPY@li.107
0.8%	0.485	0.007	--	3000.002	500	sweepx1_.ACC_COPY@li.104
0.8%	0.477	0.007	--	3000.002	500	sweepz_.ACC_COPY@li.150
0.4%	0.250	0.016	--	1503.174	500	vhone_.ACC_COPY@li.251
0.0%	0.005	0.005	6.012	--	1	vhone_.ACC_COPY@li.205
0.0%	0.001	0.000	--	6.012	1	vhone_.ACC_COPY@li.283
0.0%	0.001	0.000	--	5.000	1	vhone_.ACC_COPY@li.266

Differences in runtime

All MPI on 4096 cores

43.01 seconds

Hybrid 256 nodesx16 threads

45.05 seconds

Rest Hybrid 256x16 threads

47.58 seconds

OpenACC 256xgpu

105.92 seconds

Task 4 Fine tuning of accelerated program

- Tools that will be needed:
 - Compiler feedback on parallelization and vectorization of input application
 - Hardware counter information from the accelerator to identify bottlenecks in the execution of the application.
 - Information on memory utilization
 - Information on performance of SIMT units

Several other vendors are supplying similar performance gathering tools

Useful tools contd.

- Craypat profiling
 - Tracing: "pat_build -u <executable>" (can do APA sampling first)
 - "pat_report -O accelerator <.xf file>"; -T also useful
 - Other pat_report tables (as of perftools/5.2.1.7534)
 - acc_fu flat table of accelerator events
 - acc_time call tree sorted by accelerator time
 - acc_time_fu flat table of accelerator events sorted by accelerator time
 - acc_show_by_ct regions and events by calltree sorted alphabetically

Run and gather runtime statistics

Table 1: Profile by Function Group and Function

Time %	Time	Imb. Time	Imb. Time %	Calls	Group	Function
						PE='HIDE'
						Thread='HIDE'
100.0%	83.277477	--	--	851.0	Total	

51.3%	42.762837	--	--	703.0	ACCELERATOR	

18.8%	15.672371	1.146276	7.3%	20.0	recolor_.SYNC_COPY@li.790	←not good
16.3%	13.585707	0.404190	3.1%	20.0	recolor_.SYNC_COPY@li.793	←not good
7.5%	6.216010	0.873830	13.1%	20.0	lbm3d2p_d_.ASYNC_KERNEL@li.116	
1.6%	1.337119	0.193826	13.5%	20.0	lbm3d2p_d_.ASYNC_KERNEL@li.119	
1.6%	1.322690	0.059387	4.6%	1.0	lbm3d2p_d_.ASYNC_COPY@li.100	
1.0%	0.857149	0.245369	23.7%	20.0	collisionb_.ASYNC_KERNEL@li.586	
1.0%	0.822911	0.172468	18.5%	20.0	lbm3d2p_d_.ASYNC_KERNEL@li.114	
0.9%	0.786618	0.386807	35.2%	20.0	injection_.ASYNC_KERNEL@li.1119	
0.9%	0.727451	0.221332	24.9%	20.0	lbm3d2p_d_.ASYNC_KERNEL@li.118	

Keep data on the accelerator with acc_data region

```

!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$acc& ci12,ci13,ci14,r,b,uxyz,cell,rho,grad,index_max,index,&
!$acc& ciy,ciz,wet,np,streaming_sbuf1, &
!$acc& streaming_sbuf1,streaming_sbuf2,streaming_sbuf4,streaming_sbuf5,&
!$acc& streaming_sbuf7s,streaming_sbuf8s,streaming_sbuf9n,streaming_sbuf10s,&
!$acc& streaming_sbuf11n,streaming_sbuf12n,streaming_sbuf13s,streaming_sbuf14n,&
!$acc& streaming_sbuf7e,streaming_sbuf8w,streaming_sbuf9e,streaming_sbuf10e,&
!$acc& streaming_sbuf11w,streaming_sbuf12e,streaming_sbuf13w,streaming_sbuf14w, &
!$acc& streaming_rbuf1,streaming_rbuf2,streaming_rbuf4,streaming_rbuf5,&
!$acc& streaming_rbuf7n,streaming_rbuf8n,streaming_rbuf9s,streaming_rbuf10n,&
!$acc& streaming_rbuf11s,streaming_rbuf12s,streaming_rbuf13n,streaming_rbuf14s,&
!$acc& streaming_rbuf7w,streaming_rbuf8e,streaming_rbuf9w,streaming_rbuf10w,&
!$acc& streaming_rbuf11e,streaming_rbuf12w,streaming_rbuf13e,streaming_rbuf14e, &
!$acc& send_e,send_w,send_n,send_s,recv_e,recv_w,recv_n,recv_s)
do ii=1,ntimes
  o o o
  call set_boundary_macro_press2
  call set_boundary_micro_press
  call collisiona
  call collisionb
  call recolor

```

Now when we do communication we have to update the host

```

!$acc parallel_loop private(k,j,i)
  do j=0,local_ly-1
    do i=0,local_lx-1
      if (cell(i,j,0)==1) then
        grad (i,j,-1) = (1.0d0-wet)*db*press
      else
        grad (i,j,-1) = db*press
      end if
      grad (i,j,lz) = grad(i,j,lz-1)
    end do
  end do
!$acc end parallel_loop
!$acc update host(grad)
  call mpi_barrier(mpi_comm_world,ierr)
  call grad_exchange
!$acc update device(grad)
  
```

But we would rather not send the entire grad array back – how about